**Better Understanding AMD's Heterogeneous Interface for Portability (HIP) and How It Compares to Nvidia's Compute Unified Device Architecture (CUDA)**

*Jason Liu*

Winter 2025

# I. Introduction

Within the support of modern AI Development, an integral strut of this concerns the usage of Graphics Processing Units, commonly referred to as GPUs, which function as hardware accelerators to process large amounts of numeric operations. Initially designed to render computer graphics, GPU architecture is capable of large amounts of parallel computation, all while interfacing with various independently shared memory structures, leading to high amounts of scalability. As a result, for any large AI project, our ability to rapidly develop and test various configurations is dependent on whether we're able to effectively deploy large clusters of GPUs and efficiently schedule computationally heavy workloads in order to accelerate the task.

While GPUs do have their own assembly languages (known as PTX in Nvidia's hardware and GCN in AMD's), writing code for them requires dictating every granular interaction within the physical hardware, so any small change in our overall process often requires us to entirely rewrite most of our code. To remedy this, high-level protocols do exist. Runtime APIs provide an abstraction layer to flexibly standardize common low-level hardware specific command sequences used for memory allocation and kernel task management, and kernel languages accurately specify sets of instructions that will be executed throughout the system's hardware.

Due to their out-of-the-box support on their recent hardware, Nvidia's Compute Unified Device Architecture (CUDA) platform has generally been the industry standard, but AMD's

Heterogeneous Interface for Portability (HIP), as part of their Radeon Open Compute module (ROCm) software stack, is seeing rapid improvement and promises some significant advantages.

## II. Methodology and Challenges

In order to best understand the design philosophy and approach of each runtime API and kernel language within their respective ecosystem, tests were conducted by installing and performing the implementation of each, taken from the perspective of a new user, until reaching an intermediate level. Due to each ecosystem's much different timeline and approach to software support, the most important cases to examine reflect the drawbacks often mentioned for each:

A. *How difficult is it to apply HIP and ROCm to early-cycle technology?*

B. *Are there any hurdles with using the CUDA toolkit for late-cycle technology?*

While AMD's documentation appears overwhelming at times, it ultimately provides necessary background knowledge and correctly positions the user as the developer, for the structure of HIP is much more flexible and modular than that of CUDA. Windows support is standard for HIP SDK, but up until a few weeks ago, ROCm wasn't officially supported yet. These are separate, so while one can always program and compile the code, being able to run the compiled binary output is actually far from guaranteed. However, the open-source and flexible nature of the software made it just doable to manually assemble all the required headers and dependencies, trace and rectify the path of all the includes, and translate many of the needed calls to the console. Because the ROCm software stack is nothing more than packages and dependencies structured to be called via console, it's quite flexible and able to be adapted as needed. Additionally, once it's constructed correctly, reproducing it is as simple as cloning the file paths, making it easy to be consistently deployed across identical hardware.

Although the ROCm ecosystem places all the legwork that comes with adapting a standardized way to interpret instructions in its handling of the compiled binary, CUDA insists on organizing and stardizing these dependencies much before, integral to the build of NVCC itself. Therefore, while the CUDA toolkit often installs without any hitches, when it does fail to do so, it's nearly impossible to troubleshoot. Upon attempting to use last-generation Nvidia hardware with the appropriate backdated CUDA toolkit installation distribution, running build.py downloads and version controls all necessary dependencies, which attempts to save the user the hassle of gathering or managing these themselves. However, any inconsistencies within online repositories, like PyPI, would trigger massive issues as it attempts to install and version control each package on the spot, taking drastic corrective measures when needed, which can lead to infinite loops or massive gaps in dependencies. In this case, rectifying this issue proved impossible without extensive amounts of external knowledge as it required a near complete rewrite of multiple functions within the main file of the source distribution.

The actual code syntax for both languages is nearly identical, but there are many more resources surrounding CUDA, given that it has long been considered the standard. However, HIP becomes considerably easier to learn upon considering that one could use HIPIFY to convert large amounts of existing CUDA code into HIP, and the much newer developer tools prove miles ahead in being able to effectively troubleshoot code.

## III. Considerations and Expected Outcomes

While converting to use HIP may require adapting many current programs and working through building stable installations, the software design itself is built for much more longevity. Adapting code is relatively easy, given the improved effectiveness of newer HIPIFY tools, which are even capable of correcting syntax errors within existing CUDA code. Since HIP is

hardware-agnostic, a company using it would be free to upgrade their existing hardware without requiring changes to their software. They would also no longer be locked into a certain hardware specification or vendor, instead being able to get whatever best suits their ever-changing needs. For example, while CUDA code only works for 32-wide SIMD architecture, HIP remains flexible, allowing for the use of newer 64-wide SIMD architecture, which benefits from higher levels of parallelism for certain tasks. Additionally, code written in HIP can also be shared and applied easily without requiring any specific hardware to run it, improving collaboration or allowing for unified codebases. However, this all must come with a slight initial hit to both optimization and support within early implementations as the industry would need to adapt towards HIP being the standard. ROCm is still imperfect, so companies would have to be willing to deal with some initial friction and grow alongside it. Overall, the adoption of HIP can help break the vendor lock-in that Nvidia currently has, encouraging hardware innovation and lowering the barrier of entry for smaller players within the AI space as a whole. This would be a more healthy and sustainable direction for AI innovators as a whole.

## IV. Alternatives

Rejecting this advancement and continuing to use CUDA does prove easier within the short term. However, allowing for significant vendor lock-in, along with the rising prices and frequent issues with availability, is likely to eventually force a much more drastic decision.

Tensor-specific processors are currently under development by various companies. However, they are currently designed only for very specific use cases, making them not flexible enough to be used within most general Machine Learning tasks yet.

OpenCL is another heterogeneous computing solution for GPUs, but it only came 2 years after CUDA, meaning it’s quite antiquated and hasn’t seen any major improvements recently.

## V. Resources Utilized


- https://rocm.docs.amd.com/projects/HIP/en/latest/
- https://docs.nvidia.com/cuda/cuda-runtime-api/driver-vs-runtime-api.html#:~:text=The%20runtime%20API%20eases%20device,that%20the%20driver%20API%20has.
- https://developer.nvidia.com/cuda-downloads
- https://futhark-lang.org/blog/2024-07-17-opencl-cuda-hip.html
- https://www.techpowerup.com/forums/threads/meta-will-acquire-350-000-h100-gpus-worth-more-than-10-billion-us-dollars.318035/
- https://semianalysis.com/2024/12/22/mi300x-vs-h100-vs-h200-benchmark-part-1-training/
- https://www.tomshardware.com/pc-components/gpus/amd-mi300x-performance-compared-with-nvidia-h100
- https://stackoverflow.com/questions/4660974/how-to-create-or-manipulate-gpu-assembler?newreg=e31519279ce949f087df6322dbf2bf4d
- https://stackoverflow.com/questions/1442921/how-un-portable-is-assembly-language-really
- https://github.com/ROCm/ROCm/issues/892
- https://docs.nvidia.com/cuda/cublas/


Test Benches for various Trials were Constructed using various Consumer Hardware Configs.:

Trial Scenario A contained an AMD Ryzen 5 5600X with a Radeon RX 6800,

Trial Scenario B contained an Intel i7 6700k with a GeForce GTX 1070

* temperatures were controlled and adequate RAM was provided, all hardware ran base clock